

Sprawozdania
Wojciech Wnuk
IO 7.15

Zaawansowane programowanie w Javie
VII semestr
Rok akademicki: 2023/24

LABORATORIUM 1. STRUMIENIE I PLIKI

Zadanie 1.1. Operacje na strumieniach

```
Main.java x
1 package com.company;
2
3 import java.io.*;
4 import java.util.*;
5 import java.util.stream.*;
6
7 public class Main {
8     public static void main(String[] args) {
9
10         // Zadanie 1.1: Operacje na strumieniach
11
12         List<String> przedmioty = new ArrayList<>();
13         przedmioty.addAll(Arrays.asList("Integracja systemow", "Interakcja czlowiek-komputer",
14             "Programowanie aplikacji mobilnych na platforme Android",
15             "Programowanie aplikacji mobilnych na platforme iOS"));
16
17         Stream<String> strumienPrzedmiotow = przedmioty.stream();
18         List<Integer> listaOczen = strumienPrzedmiotow
19             .filter(przedmiot -> !przedmiot.contains("Zaaw"))
20             .map(przedmiot -> {
21                 return (int) (Math.random() * 4) + 2;
22             })
23             .collect(Collectors.toList());
24
25         listaOczen.forEach(ocena -> System.out.println("1.1: Ocena: " + ocena));
26
27         Map<Integer, Long> liczbaPowtorzen = listaOczen.stream()
28             .collect(Collectors.groupingBy(Integer::intValue, Collectors.counting()));
29
30         liczbaPowtorzen.forEach((ocena, liczba) -> {
31             if (liczba > 1) {
32                 System.out.println("Ocena " + ocena + " powtarza się " + liczba + " razy.");
33             }
34         });
35     }
```

Zadanie 1.2. Operacje na plikach

```
5
6 // Zadanie 1.2: Operacje na plikach
7
8 List<String> ocenyIZapis = listaOcen
9     .stream()
0     .map(ocena -> "Ocena: " + ocena)
1     .collect(Collectors.toList());
2
3 Collections.sort(ocenyIZapis);
4
5 try {
6     BufferedWriter writer = new BufferedWriter(new FileWriter(fileName: "oceny.txt"));
7     for (String ocena : ocenyIZapis) {
8         writer.write(ocena);
9         writer.newLine();
0     }
1     writer.close();
2 } catch (IOException e) {
3     e.printStackTrace();
4 }
5
```

Zadanie 1.3. Operacje na plikach

```
// Zadanie 1.3: Operacje na plikach

List<String> ocenyIZPliku = new ArrayList<>();

try {
    BufferedReader reader = new BufferedReader(new FileReader("oceny.txt"));
    String linia;
    while ((linia = reader.readLine()) != null) {
        ocenyIZPliku.add(linia);
    }
    reader.close();
} catch (IOException e) {
    e.printStackTrace();
}

double srednia = ocenyIZPliku.stream()
    .mapToDouble(ocena -> Integer.parseInt(ocena.substring(7)))
    .average()
    .orElse(0);

Optional<String> najlepszaOcena = ocenyIZPliku.stream()
    .max(Comparator.comparingInt(ocena -> Integer.parseInt(ocena.substring(7))));
Optional<String> najgorszaOcena = ocenyIZPliku.stream()
    .min(Comparator.comparingInt(ocena -> Integer.parseInt(ocena.substring(7))));

System.out.println("Średnia ocen: " + srednia);
System.out.println("Najlepsza ocena: " + najlepszaOcena.orElse("Brak ocen"));
System.out.println("Najgorsza ocena: " + najgorszaOcena.orElse("Brak ocen"));
}
```

Wyniki działania kodu:

```
"C:\Program Files\Java\jdk-17\bin
1.1: Ocena: 4
1.1: Ocena: 3
1.1: Ocena: 5
1.1: Ocena: 4
Ocena 4 powtarza się 2 razy.
Średnia ocen: 4.0
Najlepsza ocena: Ocena: 5
Najgorsza ocena: Ocena: 3
```

```
oceny — Notatnik
Plik  Edycja  Format  Widok  F
Ocena: 3
Ocena: 4
Ocena: 4
Ocena: 5
```

Wnioski:

Wykorzystanie strumieni w Javie pozwala na wygodne i efektywne przetwarzanie danych, zarówno w przypadku operacji na kolekcjach, jak i operacji na plikach. Funkcje takie jak filter, map i collect są użytecznymi narzędziami do manipulacji danymi w strumieniach, umożliwiając filtrowanie, transformację i zbieranie wyników w odpowiednich strukturach danych.

LABORATORIUM 2. WYKORZYSTANIE PLIKÓW XML.

Zadanie 2.1. Tworzenie XML

```
Main.java × students.xml × Student.java ×
1 package com.company;
2 public class Student {
3     private String name;
4     private int age;
5     private String course;
6
7     public Student(String name, int age, String course) {
8         this.name = name;
9         this.age = age;
10        this.course = course;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28
29    public String getCourse() {
30        return course;
31    }
32
33    public void setCourse(String course) {
34        this.course = course;
35    }
36 }
```

```
public class Main {  
  
    public static void main(String[] args) {  
        List<Student> students = new ArrayList<>();  
        students.add(new Student( name: "John Doe", age: 20, course: "Computer Science"));  
        students.add(new Student( name: "Jane Smith", age: -3, course: "Mathematics"));  
        students.add(new Student( name: "Bob Johnson", age: 22, course: "Physics"));  
  
        createXML(students, filename: "students.xml");  
        List<Student> parsedStudents = parseXML( filename: "students.xml");  
  
        for (Student student : parsedStudents) {  
            if (student != null) {  
                System.out.println("Name: " + student.getName());  
                System.out.println("Age: " + student.getAge());  
                System.out.println("Course: " + student.getCourse());  
                System.out.println();  
            } else {  
                System.out.println("Nie można utworzyć obiektu Student");  
            }  
        }  
    }  
}
```



```

public static void createXML(List<Student> students, String filename) {
    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        Document doc = docBuilder.newDocument();
        Element rootElement = doc.createElement( tagName: "Students");
        doc.appendChild(rootElement);

        for (Student student : students) {
            Element studentElement = doc.createElement( tagName: "Student");
            rootElement.appendChild(studentElement);

            Element nameElement = doc.createElement( tagName: "Name");
            nameElement.appendChild(doc.createTextNode(student.getName()));
            studentElement.appendChild(nameElement);

            Element ageElement = doc.createElement( tagName: "Age");
            ageElement.appendChild(doc.createTextNode(String.valueOf(student.getAge())));
            studentElement.appendChild(ageElement);

            Element courseElement = doc.createElement( tagName: "Course");
            courseElement.appendChild(doc.createTextNode(student.getCourse()));
            studentElement.appendChild(courseElement);
        }

        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File(filename));
        transformer.transform(source, result);

        System.out.println("XML file saved as " + filename);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Zadanie 2.2. Odczyt XML

```
public class Main {

    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student( name: "John Doe", age: 20, course: "Computer Science"));
        students.add(new Student( name: "Jane Smith", age: -3, course: "Mathematics"));
        students.add(new Student( name: "Bob Johnson", age: 22, course: "Physics"));

        createXML(students, filename: "students.xml");
        List<Student> parsedStudents = parseXML( filename: "students.xml");

        for (Student student : parsedStudents) {
            if (student != null) {
                System.out.println("Name: " + student.getName());
                System.out.println("Age: " + student.getAge());
                System.out.println("Course: " + student.getCourse());
                System.out.println();
            } else {
                System.out.println("Nie można utworzyć obiektu Student");
            }
        }
    }
}
```

```

public static List<Student> parseXML(String filename) {
    List<Student> students = new ArrayList<>();

    try {
        File xmlFile = new File(filename);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);

        doc.getDocumentElement().normalize();

        NodeList studentNodes = doc.getElementsByTagName("Student");

        for (int i = 0; i < studentNodes.getLength(); i++) {
            Node studentNode = studentNodes.item(i);
            if (studentNode.getNodeType() == Node.ELEMENT_NODE) {
                Element studentElement = (Element) studentNode;
                String name = studentElement.getElementsByTagName("Name").item(0).getTextContent();
                String ageStr = studentElement.getElementsByTagName("Age").item(0).getTextContent();
                String course = studentElement.getElementsByTagName("Course").item(0).getTextContent();
                int age;

                try {
                    age = Integer.parseInt(ageStr);
                } catch (NumberFormatException e) {
                    age = -1;
                }

                if (!name.isEmpty() && age > 0 && !course.isEmpty()) {
                    students.add(new Student(name, age, course));
                } else {
                    students.add(null);
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return students;
}

```

Zadanie 2.3. Filtracja XML

```

public static List<Student> parseXML(String filename) {
    List<Student> students = new ArrayList<>();

    try {
        File xmlFile = new File(filename);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);

        doc.getDocumentElement().normalize();

        NodeList studentNodes = doc.getElementsByTagName("Student");

        for (int i = 0; i < studentNodes.getLength(); i++) {
            Node studentNode = studentNodes.item(i);
            if (studentNode.getNodeType() == Node.ELEMENT_NODE) {
                Element studentElement = (Element) studentNode;
                String name = studentElement.getElementsByTagName("Name").item(0).getTextContent();
                String ageStr = studentElement.getElementsByTagName("Age").item(0).getTextContent();
                String course = studentElement.getElementsByTagName("Course").item(0).getTextContent();
                int age;

                try {
                    age = Integer.parseInt(ageStr);
                } catch (NumberFormatException e) {
                    age = -1;
                }

                if (!name.isEmpty() && age > 0 && !course.isEmpty()) {
                    students.add(new Student(name, age, course));
                } else {
                    students.add(null);
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return students;
}

```

```

for (Student student : parsedStudents) {
    if (student != null) {
        System.out.println("Name: " + student.getName());
        System.out.println("Age: " + student.getAge());
        System.out.println("Course: " + student.getCourse());
        System.out.println();
    } else {
        System.out.println("Nie można utworzyć obiektu Student");
    }
}
}

```

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <Students>
3   <Student>
4     <Name>John Doe</Name>
5     <Age>20</Age>
6     <Course>Computer Science</Course>
7   </Student>
8   <Student>
9     <Name>Bob Johnson</Name>
10    <Age>22</Age>
11    <Course>Physics</Course>
12  </Student>
13  <Student>
14    <Name>Jane Smith</Name>
15    <Age>-3</Age>
16    <Course>Mathematics</Course>
17  </Student>
18 </Students>
```

Wnioski:

Praca z danymi w formacie XML jest powszechna w dzisiejszych aplikacjach. Pozwala na przechowywanie, wymianę i analizę danych w sposób strukturalny. Mechanizmy filtrowania pozwalają na sprawdzenie czy dane spełniają określone kryteria, co jest istotne w przypadku danych, które nie zawsze są kompletnie zgodne lub poprawne.

LABORATORIUM 6. TWORZENIE APLIKACJI Z WYKORZYSTANIEM BIBLIOTEKI LOMBOK.

Zadanie 6.1. Klasa modelowa – bez użycia biblioteki Lombok

```
Main.java × Employee.java ×
1 package com.company;
2
3 public class Employee {
4     private String name;
5     private Integer salary;
6     private Integer age;
7     private String role;
8
9     public Employee(String name, Integer salary, Integer age, String role) {
10         this.name = name;
11         this.salary = salary;
12         this.age = age;
13         this.role = role;
14     }
15
16     public Employee() {
17     }
18
19     public String getName() {
20         return name;
21     }
22
23     public void setName(String name) {
24         this.name = name;
25     }
26
27     public Integer getSalary() {
28         return salary;
29     }
30
31     public void setSalary(Integer salary) {
32         this.salary = salary;
33     }
34
35     public Integer getAge() {
36         return age;
37     }
38
39     public void setAge(Integer age) {
40         this.age = age;
41     }
```

```
Main.java x Employee.java x
1 package com.company;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         Employee employee1 = new Employee( name: "John", salary: 50000, age: 30, role: "Manager");
8         Employee employee2 = new Employee( name: "Jane", salary: 60000, age: 35, role: "CTO");
9         Employee employee3 = new Employee( name: "Jack", salary: 40000, age: 25, role: "Developer");
10
11
12
13     }
```

```
Main.java x Employee.java x
package com.company;

3 related problems
public class Employee {
    private String name;
    private Integer salary;
    private Integer age;
    private String role;
    private Integer DaysOffWork;

    3 related problems
    public Employee(String name, Integer salary, Integer age, String role, Integer daysOffWork) {
        this.name = name;
        this.salary = salary;
        this.age = age;
        this.role = role;
        DaysOffWork = daysOffWork;
    }

    3 related problems
    public Employee() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getSalary() {
        return salary;
    }

    public void setSalary(Integer salary) {
        this.salary = salary;
    }

    public Integer getAge() {
        return age;
    }
}
```

Terminal

Zadanie 6.2. Klasa modelowa – z biblioteką Lombok

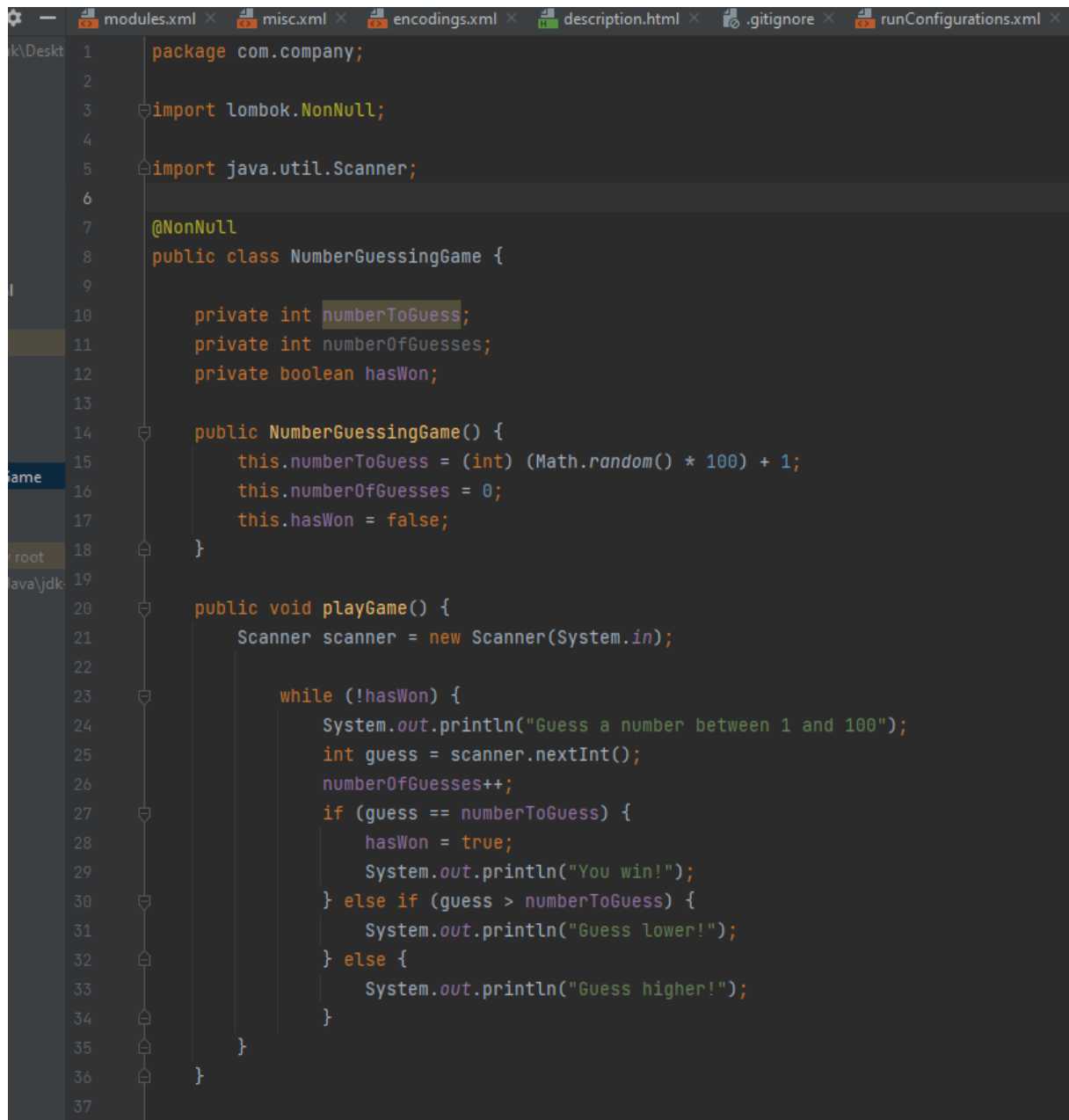
```
workspace.xml × runConfigurations.xml × project-template.xml  
package com.company;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter @Setter @NoArgsConstructor @AllArgsConstructor  
public class Employee {  
    private String name;  
    private Integer salary;  
    private Integer age;  
    private String role;  
    private Integer DaysOffWork;  
}
```

```
workspace.xml × runConfigurations.xml × project-template.xml × E  
package com.company;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter @Setter @NoArgsConstructor @AllArgsConstructor  
public class Employee {  
    private String name;  
    private Integer salary;  
    private Integer age;  
    private String role;  
  
}
```

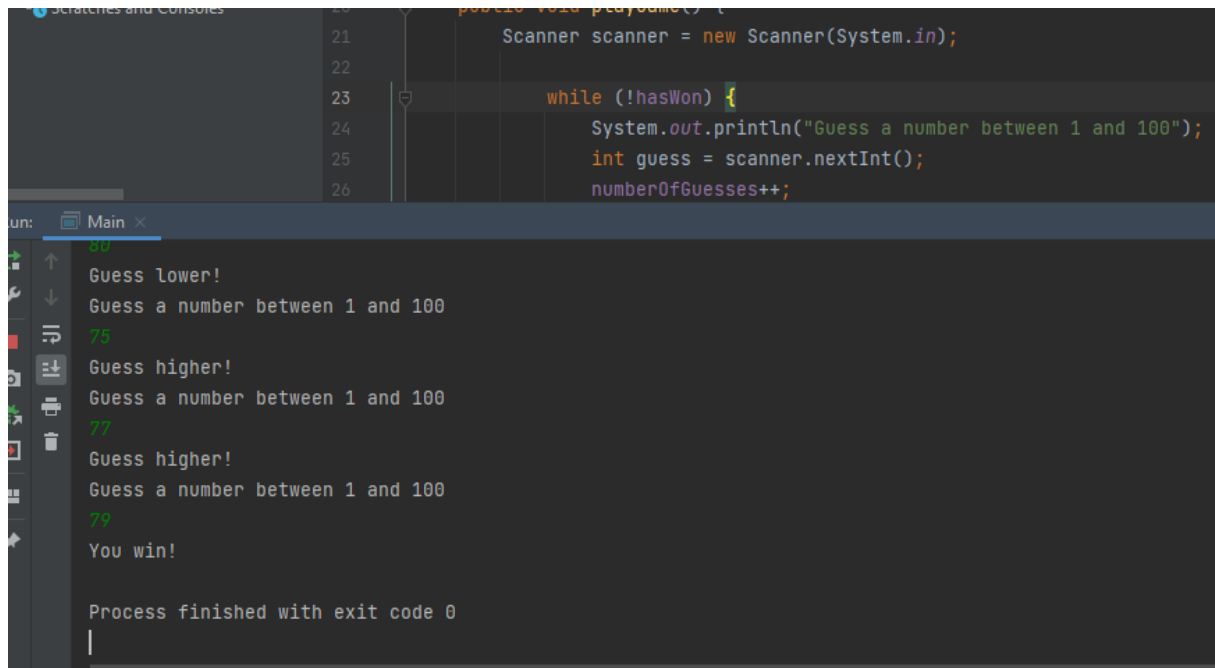
Czy po zmianie liczby pól w klasie modelowej należy coś zmieniać?

Nic nie trzeba zmieniać, Lombok sam aktualizuje na bieżąco gettery, settery oraz konstruktory.

Zadanie 6.3. Dane od użytkownika

A screenshot of an IDE window showing a Java file named 'NumberGuessingGame.java'. The code is written in a dark-themed editor. The package is 'com.company'. It imports 'lombok.NonNull' and 'java.util.Scanner'. The class 'NumberGuessingGame' is annotated with '@NonNull' and contains three private fields: 'numberToGuess' (int), 'numberOfGuesses' (int), and 'hasWon' (boolean). The constructor 'NumberGuessingGame()' initializes 'numberToGuess' to a random number between 1 and 100, 'numberOfGuesses' to 0, and 'hasWon' to false. The 'playGame()' method uses a 'Scanner' to read user input in a loop until the user wins or guesses correctly. The IDE interface shows several tabs at the top: 'modules.xml', 'misc.xml', 'encodings.xml', 'description.html', '.gitignore', and 'runConfigurations.xml'. The left sidebar shows a project tree with 'root' and 'java\jdk' folders. The line numbers on the left range from 1 to 37.

```
1 package com.company;
2
3 import lombok.NonNull;
4
5 import java.util.Scanner;
6
7 @NonNull
8 public class NumberGuessingGame {
9
10     private int numberToGuess;
11     private int numberOfGuesses;
12     private boolean hasWon;
13
14     public NumberGuessingGame() {
15         this.numberToGuess = (int) (Math.random() * 100) + 1;
16         this.numberOfGuesses = 0;
17         this.hasWon = false;
18     }
19
20     public void playGame() {
21         Scanner scanner = new Scanner(System.in);
22
23         while (!hasWon) {
24             System.out.println("Guess a number between 1 and 100");
25             int guess = scanner.nextInt();
26             numberOfGuesses++;
27             if (guess == numberToGuess) {
28                 hasWon = true;
29                 System.out.println("You win!");
30             } else if (guess > numberToGuess) {
31                 System.out.println("Guess lower!");
32             } else {
33                 System.out.println("Guess higher!");
34             }
35         }
36     }
37 }
```



The screenshot shows an IDE with two panels. The top panel displays Java code for a number-guessing game. The code includes a `Scanner` object, a `while` loop that prompts the user to guess a number between 1 and 100, and a `numberOfGuesses` counter. The bottom panel shows the execution output for the `Main` class. The output shows the user making four incorrect guesses (50, 75, 77, 79) and finally winning with the guess 70. The process finished with exit code 0.

```
21 Scanner scanner = new Scanner(System.in);
22
23 while (!hasWon) {
24     System.out.println("Guess a number between 1 and 100");
25     int guess = scanner.nextInt();
26     numberOfGuesses++;
27 }
```

Run: Main ×

50
Guess lower!
Guess a number between 1 and 100
75
Guess higher!
Guess a number between 1 and 100
77
Guess higher!
Guess a number between 1 and 100
79
You win!

Process finished with exit code 0

Wnioski:

Lombok znacznie przyspiesza nam zmiany dokonywane w klasach modelowych, automatycznie gdy zmieniamy pola generuje nam odpowiednie metody. Dodatkowo znacznie skraca ilość kodu oraz zapewnia większą czytelność.

